Amendments to the Specification:

Please replace the title on page 1 with the following amended title:

ARRAY SEARCHING OPERATIONS WHICH FETCH AND COMPARE MULTIPLE ARRAY ELEMENTS

Please replace the paragraph beginning at page 1, line 16 with the following amended paragraph:

Figure 1 is a block diagram illustrating an example of a pipelined programmable processor ~~according to the invention~~.

Please replace the paragraph beginning at page 1, line 21 with the following amended paragraph:

Figure 3 is a flowchart for implementing an example array manipulation machine instruction ~~according to the invention~~.

Please add the following new paragraphs at page 2, line 1:

Figure 5 is a flowchart for a single SEARCH instruction.

Figure 6 is a flowchart where a software application issues N/M SEARCH instructions and, upon completion of the N/M SEARCH instructions, determines an extreme value for an entire array.

Please replace the paragraph beginning at page 2, line 17 with the following amended paragraph:

Control unit 6 controls the flow of instructions and data through the various stages of pipeline 4. During the processing of an instruction, for example, control unit 6 directs the various components of the ~~pipelined~~ pipeline 4 to fetch and decode the instruction, perform the corresponding operation and write the results back to memory or local registers.

Please replace the paragraph beginning at page 4, line 1 with the following amended paragraph:

Generally, the sequence of SEARCH instructions allows ~~allow~~ the processor 2 to process M sets of elements in parallel to identify an "extreme value", such as a maximum or a minimum, for each set. During the execution of the search instructions, processor 2 stores references to the location of the extreme value of each of the M sets of elements. Upon completion of the N/M instructions, as described in detail below, the software application analyzes the references to the extreme values for each set to quickly identify an extreme value for the array. For example, the search instruction allows the software applications to quickly identify either the first or last occurrence of a maximum or minimum value. Furthermore, as explained in detail below, processor 2 implements the operation in a fashion suitable for vectorizing in a pipelined processor across the M execution units 25.

Please replace the paragraph beginning at page 4, line 17 with the following amended paragraph:

As described above, a software application searches an array of data by issuing N/M SEARCH machine instructions to processor 2. Figure 3 is a flowchart illustrating an example mode of operation ~~20~~ 300 for processor 2 when it receives a single SEARCH machine instruction. Process ~~20~~ 300 is described with reference to identifying the last occurrence of a minimum value within the array of elements; however, process ~~20~~ 300 can be easily modified to perform other functions such as identifying the first occurrence of a minimum value, the first occurrence of a maximum value or a last occurrence of a maximum value.

Please replace the paragraph beginning at page 5, line 1 with the following amended paragraph:

For exemplary purposes, process ~~20~~ 300 is described in assuming M equals 2, i.e., processor 2 concurrently processes two sets of elements, each set having N/2 elements. However, the process is not limited as such and is readily extensible to concurrently process more than two sets of elements. In general, process ~~20~~ 300 facilitates vectorization of the search process by fetching pairs of elements as a single data quantity and processing the element pairs through pipeline 4 in parallel, thereby reducing the total number of clock cycles necessary to identify the minimum value within the array. Although applicable to other architectures, process ~~20~~ 300 is well suited for a pipelined processor 2 having multiple execution units in the EX stage. For ~~each set~~ the two sets of elements, process ~~20~~ 300 maintains two pointer registers, P_{Even} and P_{Odd}, that store locations for the current extreme value within the corresponding set. In addition, process ~~20~~ 300

Q7 Cont'd.

maintains two accumulators, A0 and A1,that hold the current extreme values for the sets. The pointer registers and the accumulators, however, may readily be implemented as general-purpose data registers without departing from process 300 ~~30~~.

Please replace the paragraph beginning at page 5, line 23 with the following amended paragraph:

Referring to Figure 3, in response to each SEARCH instruction, processor 2 fetches a pair of elements in one clock cycle as a single data quantity (301~~21~~). For example, processor 2 may fetch two adjacent 16-bit values as one 32-bit quantity. Next, processor 2 compares the even element of the pair to a current minimum value for the even elements (302~~22~~) and the odd element of the pair to a current minimum value for the odd elements (304~~24~~).

Q8

Please replace the paragraph beginning at page 6, line 4 with the following amended paragraph:

When a new minimum value for the even elements is detected, processor 2 updates accumulator A0 to hold the new minimum value and updates a pointer register P_{Even} to hold a pointer to point to a corresponding data quantity within the array (303~~23~~). Similarly, when a new minimum value for the odd elements has been detected, processor 2 updates accumulator A1 and a pointer register P_{Odd} (305~~25~~). In this example, each pointer register P_{Even} and P_{Odd} points to the data quantity and not the individual elements, although the process is not limited as such. Processor 2 repeats the process until all of the elements within the array have been processed

Q9

Q9 Cont'd. (30626). Because processor 2 is pipelined, element pairs may be fetched until the array is processed.

Please replace the paragraph beginning at page 7, line 8 with the following amended paragraph:

In a typical application, a programmer develops a software application or subroutine that issues the N/M search instructions, probably from within a loop construct. The programmer may write the software application in assembly language or in a high-level software language. A compiler is typically invoked to process ~~processes~~ the high-level software application and generate the appropriate machine instructions for processor 2, including the SEARCH machine instructions for searching the array of data.

Q10

Please replace the paragraph beginning at page 7, line 17 with the following amended paragraph:

Figure 4 is a flowchart of an example software routine 30 for invoking the example machine instructions illustrated above. First, the software routine 30 initializes the registers including initializing A0 and A1 and pointers ~~pointing~~ P_{Even} and P_{Odd} to the first data quantity within the array (31). In one embodiment, software routine 30 initializes a loop count register with the number of SEARCH instructions to issue (N/M). Next, routine 30 issues the SEARCH machine instruction N/M times (32). This can be accomplished a number of ways, such as by invoking a hardware loop construct supported by processor 2. Often, however, a

Q11

compiler may unroll a software loop into a sequence of identical SEARCH instructions (32).

Please replace the paragraph beginning at page 8, line 10 with the following amended paragraph:

Next, in order to identify the last occurrence of the minimum value for the entire array, routine 30 first increments P_{Odd} by a single element, such that P_{Odd} points directly at the minimum odd element (33). Routine 30 compares the accumulators A0 and A1 to determine whether the accumulators contain the same value, i.e., whether the minimum of the odd elements equals the minimum of the even elements (34). If so, the routine 30 compares the pointers to determine whether P_{Odd} is less than P_{Even} and, therefore, ~~P_{Odd} and P_{Even}~~ whether the minimum even value occurred earlier or later in the array (35). Based on the comparison, the routine determines whether to copy P_{Odd} into P_{Even} (37).

Please replace the paragraph beginning at page 9, line 12 with the following amended paragraph:

Figure 5 illustrates the operation for a single SEARCH instruction as generalized to the case where processor 2 is capable of processing M elements of the array in parallel, such as when processor 2 includes M execution units. The SEARCH instruction causes processor 2 to fetch M elements in a single fetch cycle (51). Furthermore, in this example, processor 2 maintains M pointer registers to store addresses (locations) of ~~a~~ corresponding extreme values ~~value~~ for ~~each of~~ the M sets of elements. After fetching the M elements,

processor 2 concurrently compares the M elements to ~~a~~ current extreme values ~~value~~ for the respective element sets ~~set~~, as stored in M accumulators (52). Based on the comparisons, processor 2 updates the M accumulators and the M pointer registers (53).

Q13 cont'd.

Please replace the paragraph beginning at page 9, line 26 with the following amended paragraph:

Figure 6 illustrates the general case where a software application issues N/M SEARCH instructions and, upon completion of the instructions, determines the extreme value for the entire array. First, the software application initializes a loop counter, the M accumulators used to store the current extreme values for the M element sets and the M pointers used to store the locations of the extreme values (61). Next, the software application issues N/M SEARCH instructions (62). After completion of the instructions, the software application may adjust each of the M pointer registers to correctly reference its respective extreme value, instead of the data quantity holding the extreme value (63). After adjusting the pointer registers, the software application compares the M extreme values for the M element sets to identify an extreme value for the entire array, i.e., a maximum value or a minimum value (64). Then, the software application may use the pointer registers to determine whether more than one of the element sets have an extreme value equal to the array extreme value and, if so, determine which extreme value occurred first, or last, depending upon the desired search function (65).

Q14
